AUTOMATIC TRAIN RUNNING INFORMATION SYSTEM

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology

in

Electronics and Instrumentation Engineering

By

Varun Sreenivas- 10507030

N. Kiran Kumar- 10507034

Under the Guidance of

Dr.S.K.Patra

Head of the Department

Department of Electronics and Communication Engineering

National Institute Of Technology, Rourkela

PIN 769008, Orissa, India

2009

ACKNOWLEDGMENT

We place on record and warmly acknowledge the continuous encouragement, invaluable supervision, timely suggestions and inspired guidance offered by our guide **Dr.S.K.Patra**, Professor (Head of the Department of Electronics and Instrumentation Engineering), National Institute of Technology, Rourkela, in bringing this report to a successful completion.

We are once again grateful to **Prof G.S. Rath, Prof G.Panda, Prof K.K.Mahapatra, Prof S.Meher, Prof T.K. Dan, Prof S.K Behera, Prof U.C Pati, Prof D.P Acharya, Prof A.K.Sahoo, Prof P.K. Sahu, and Prof P.Singh** for permitting us to make use of the facilities available in the department to carry out the project successfully. Last but not the least we express our sincere thanks to all our friends who have patiently extended all sorts of help for accomplishing this undertaking.

Finally we extend our gratefulness to one and all who are directly or indirectly involved in the successful completion of this project work.

Varun Sreenivas (10507030)

N.Kiran Kumar (10507034)

National Institute of Technology
Rourkela

CERTIFICATE

This is to certify that the thesis entitled, "Automatic train running information system" submitted by Sri Varun Sreenivas and Sri N. Kiran Kumar in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Electronics & Instrumentation Engineering at National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University / Institute for the award of any Degree or Diploma.

Date

Dr. S. K. PATRA
Head of the Department
Dept. of Electronics & Instrumentation Engg
National Institute of Technology
Rourkela – 769008

CONTENTS

Page No.

Abstract iii

Chapter 1 INTRODUCTION 1-2

1.1 Automatic Train Running Information System 2

Chapter 2 SIMULATIONS IN RAILWAY WIRELESS COMMUNICATION NETWORK 3-11

2.1 Simulation using Two Trains 4

2.2 Simulation using Multiple Trains 6

2.3 Railway Station Scenario Simulation 10

Chapter 3 USE OF GSM TECHNIQUES FOR MESSAGE TRANSMISSION 12-14

3.1 Establishing connection between Wireless Modem (SIMADO) and message transmitter (Mobile) 13

Chapter 4 WAP TO HTTP CONVERSION 15-21

4.1 Nokia Mobile Web Browser 4.0 16

4.2 Nokia WAP Gateway Simulator 4.0 19

Chapter 5 ESTABLISHMENT OF CONNECTION BETWEEN NWGS AND DATABASE 22-25

5.1 Java Code for establishing connection between MS Access Database and WAP Gateway Simulator 23

Chapter 6 CREATION OF LIVE RAILWAY TIMETABLE 26-52

6.1 Interfacing MS Access Database with Netbeans in order to generate live Railway timetable 27

6.2 Java Programs for Server, Client, Train information 29

Chapter 7 INTERFACE OF DATABASE WITH MULTIPLE TRAINS 53-56

7.1 Interfacing the Database with SMS's from multiple mobiles 54

CONCLUSION AND FUTURE SCOPE 56

REFERENCES 57

ABSTRACT

The main objective of our project is to design an automatic train running passenger information system. The first step in achieving this was to establish connections between the trains arriving and leaving the station, and the base station in the station. We were able to establish wireless communication within the station premises, between the trains and base station.

The platform on which the basic format of communication in a station scenario was simulated was Qualnet. We used the SIMADO GSM Modem in the role of base station in the station. And then we took the help of Nokia Mobile Browser and Nokia WAP Gateway Simulator to establish connection between base station and the server. We were able to update the information using WAP Gateway Simulator.

Then, we interfaced the NWGS with MS Access Database to get the HTTP information received to get it updated and maintained in a database. Then, this database was interfaced with Netbeans to generate live Railway Timetable. We used three JAVA programs each for creating server, client and train information used for this purpose. Then we made use of VBA code to connect database with multiple trains in different stations.

Chapter 1

INTRODUCTION

Topic: **Automatic train running information system**

Theory: Railway information system is generally built upon a computer based network to support rail information collection transmission, processing and dissimilation in order to ensure safe and stable rail transportation and provide high quality operational service as well as passenger information system. A new generation wireless application protocol and web technologies from next generation is utilized.

The existing railway information system uses sensors and PVC insulated Copper Cables. The sensor is mounted on the lamp-post near the station which senses the train and sends the information regarding the arrival of a train, its arrival time, train no. etc. to the receiver in the station. This receiver receives the information. The information received is updated into the local server manually. This server, which has the master clock and is interfaced with display boards and coach guidance boards, signal posts and many slave clocks in the station, informs the public through them.

In some stations, the whole information system is done manually. This system has some disadvantages such as need of manual operation, high cost factor and problems related with cables, such as attenuation and interference.

The application of the technology is as follows, in a railway station, say there are two trains arriving from different directions entering track one and track two. There are transmitters fixed inside the train. These transmitters transmit signals to the receiver in the station. In the signal send details such as train number, next stop, arrival time, departure time is present. From the receiver the information is converted from WAP to HTTP and the information is loaded into the main server where the railway timetable is maintained and is continuously updated based on the information continuously received from various trains entering and departing the station.

Chapter 2

SIMULATIONS IN RAILWAY WIRELESS COMMUNICATION NETWORK

Topic: Simulation using two trains

Platform used: Qualnet.

Theory: Qualnet Developer provides a comprehensive environment for designing protocols, creating and animating experiments, and analyzing the results of those experiments. Qualnet simulator is used to simulate the real time operation of vehicles, trains, aeroplanes etc using wireless and various other modes of communication. It has the ability to generate various scenarios of communication along with the ability to generate traffic. Qualnet simulator is used here in order to establish connection between trains as well as to establish connection between trains and the railway station.

Procedure: For simulation purpose Qualnet is used. In the scenario generator we create two transmitters that will transmit signals from two different trains entering the station. There is a wireless network icon placed in the grid that is denoted by a cloud symbol. This wireless network exists in the station premises. The two transmitters are now connected to the wireless symbol by pulling a link from each of the nodes to the wireless network. This wireless network is used for transmitting and receiving signals. The information from the wireless network is received by a receiver in the station. From node3 we convert the signal received from WAP to http format and it is loaded into the railway server. We can observe the amount of packets transferred by viewing the input and output graph icon. After the links and nodes have been inserted into the grid we need to run the scenario. The scenario can be run in a step mode where the entire scenario is run in a step manner and the transmission of the signals can be observed. The green arrow represents direction of propagation. The frequency of transmission can be set in the properties table.

Simulation: Stage 1

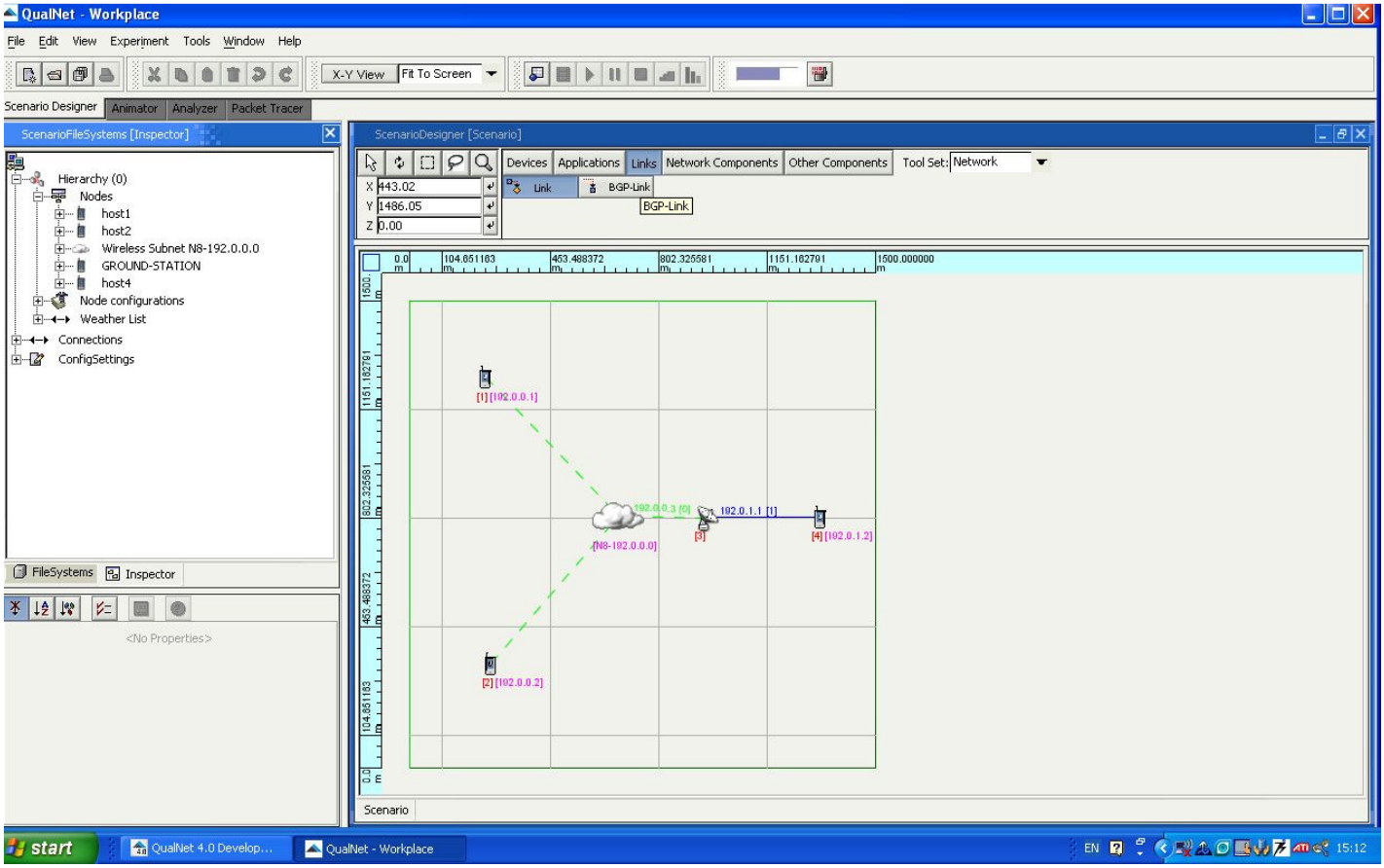

Fig.2.1: Simulation showing establishment of wireless connection between two trains and station.

Topic: Simulation using multiple trains

Theory: In this simulation 4 trains were taken into consideration. All 4 trains had different transmitters fixed in them. Train1-track1 similarly train2, 3, 4 on their corresponding tracks. All of them as soon as they enter the station premises they get linked with the wireless network present in the station. As information is send to the main server via the wireless network. There is a ground station on the main platform that receives information from various trains via the wireless network. The ground station is connected to the main hub where the wireless information is converted to http format and is uploaded to the main server. The amount of data transmitted by the train and received by the ground station has been plotted.

Procedure: In Qualnet the devices are selected (nodes). A information goes directly to the server. The scenario animator is clicked; we reduce the run speed so as to observe the signal transmission direction. Hence it is simulated and plotted wireless network depicted by cloud is used. Using wireless link depicted in green colour they are linked. In order to generate a ground station scenario designer is clicked, a node is selected, and upon right clicking there is an option to change the node to ground station. Then it is linked to a hub from where the information goes directly to the server. The scenario animator is clicked; we reduce the run speed so as to observe the signal transmission direction. Hence it is simulated and plotted.

Simulation: Stage 2

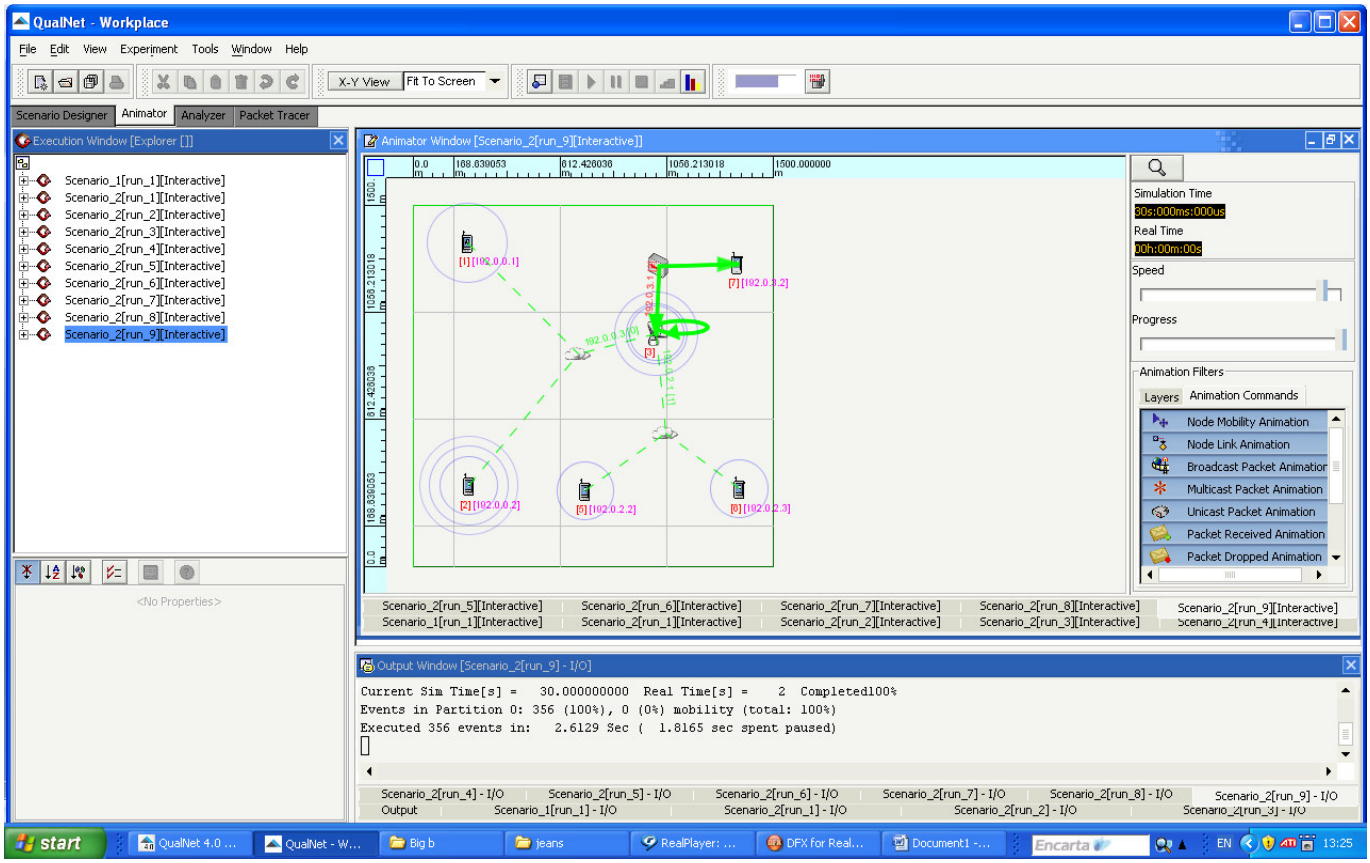

Fig.2.2: Simulation showing establishment of wireless connection between multiple trains and station.

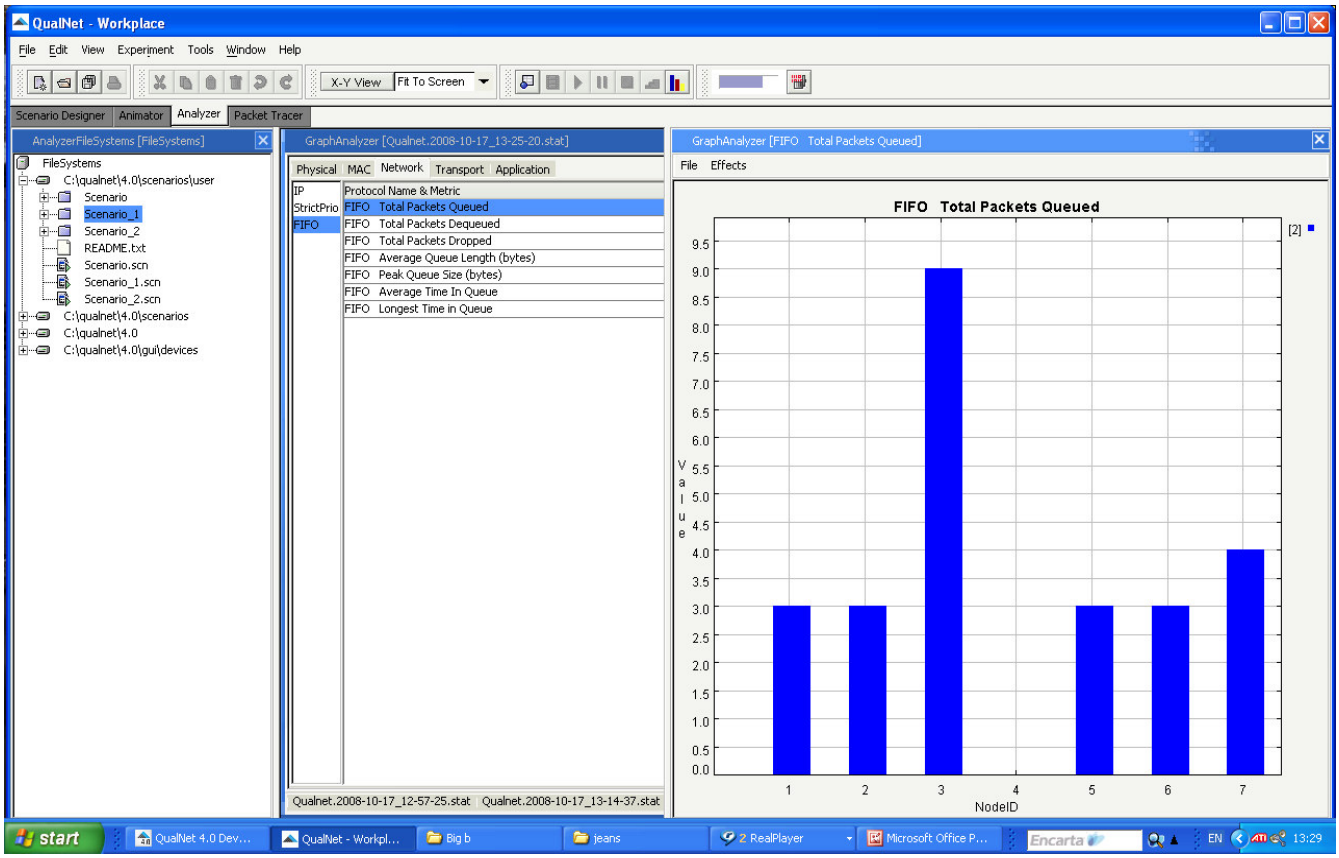

Fig.2.3: The data packets transmitted by a train.

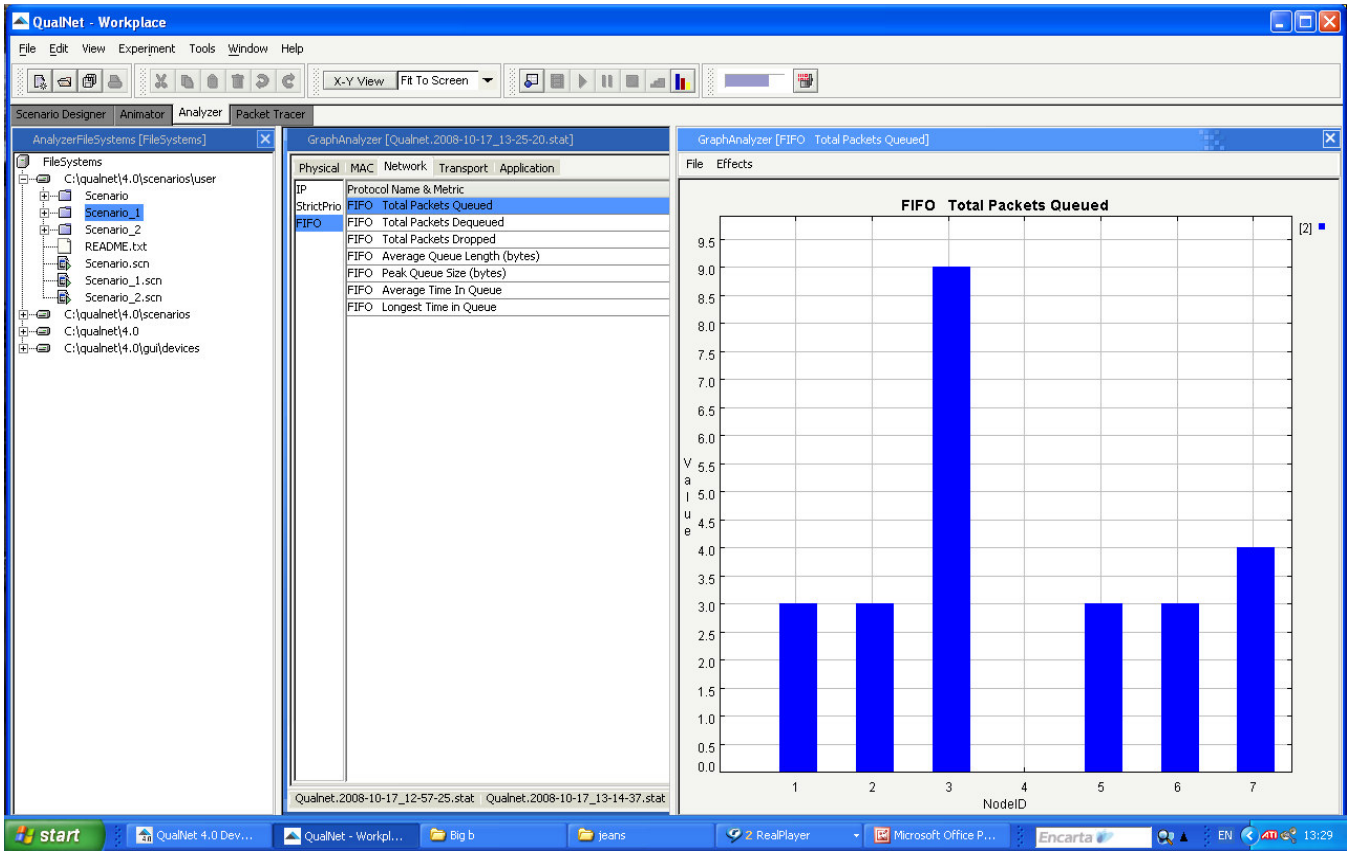

Fig.2.4: The data packets received by the station receiver.

Topic: **Railway station scenario simulation**

Platform: **Qualnet**

Procedure: A high speed scenario is used here, where several trains pass the railway station. Some stop and some are non-stop. The paths of the trains have been designed with the help of waypoint or red flags. As the trains are linked to the wireless network from where information is received by the ground station and then sent to the server. In order to make the nodes (trains) mobile, we need to click on system scenario designer and then right click on the nodes, click on the drop down mobility and select the different types of mobility patterns. Nearby roads and streets have been depicted in the scenario. Here we use the method of random waypoint which is depicted by the red flags which shows the direction of movement of the train. As the trains cross the station they transmit messages to the base station from where the message is converted from wireless application protocol to http format. Then the message is loaded onto the server in http format so that the users can access the required information from the server. In the scenario we have allocated different IP addresses to different nodes/trains so that each of them has a unique IP. The cloud like structure depicted is used to show the wireless network established between the trains and the base stations.

Simulation: Stage 3

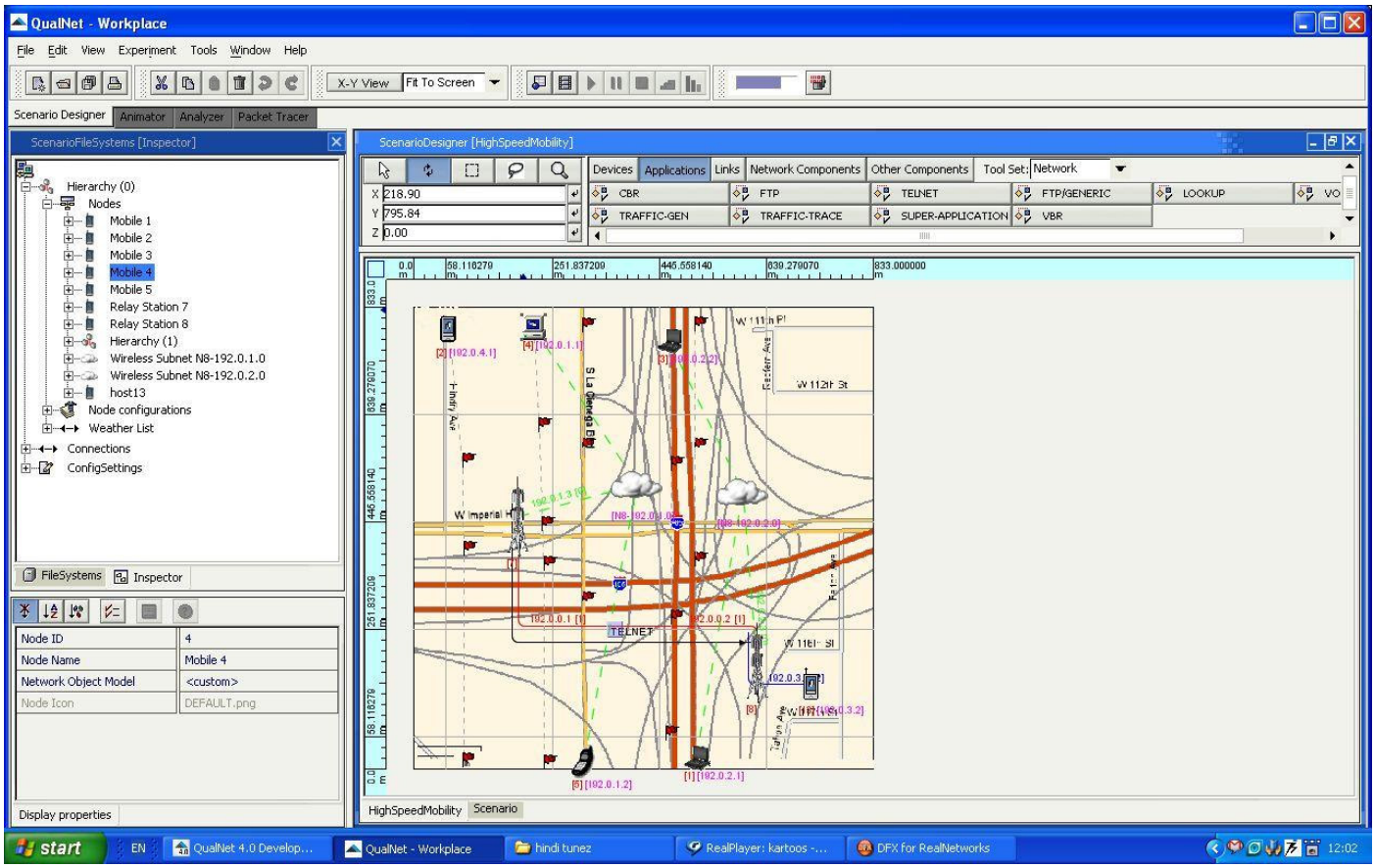

Fig.2.5: Simulation of a station scenario.

Chapter 3

USE OF GSM TECHNIQUES FOR MESSAGE TRANSMISSION

Topic: **Establishing connection between the wireless modem (SIMADO) and a message transmitter (mobile).**

Platforms: **SIMADO GSM modem, HyperTerminal (Win XP)**

Theory: GSM is a cellular network, which means that mobile phones connect to it by searching for cells in the immediate vicinity. There are five different cell sizes in a GSM network—macro, micro, pico, femto and umbrella cells. The coverage area of each cell varies according to the implementation environment.

Here we have used SIMADO modem which provides the required GSM services. SIMADO modem is a compact ready to use unit providing a standard nine pin RS232 port (serial) for data communication. When a PC is connected to the serial port the SIMADO modem can efficiently be used for web browsing, sending and receiving SMS, data transfer and fax services.

Procedure: **Wireless Communication between a train(s) entering a station and the receiver in the station:**

With the help of the SIMADO MODEM we were able to establish connection between modem and a mobile.

Text messages can be sent and received through this wireless network.

Inside the train there will be a transmitter that will transmit a text message to the receiver in the station as soon as the trains enter the station premises. The trains will transmit information such as its arrival time, departure time, train number, platform number and next stop.

Basic process: Using the HyperTerminal in the computer, we can create a new connection and enter a name for it. After that we need to set the baud rate in the properties option. Once the properties have been set the HyperTerminal window will have options to wait for call. After clicking on wait for a call, the modem will wait for a call or message from any remote transmitter. The modem will be in the connected mode. We can send messages to the modem in this way.

Train: There will be an intelligent system embedded into the train's transmission system such that it is interfaced with the train's motion system that tracks the kilometres travelled by the train. The system monitors the number of kilometres the train has moved from its source station. As the train starts its journey, there is a counter that keeps a track of the distance travelled. The distance to the next station from the starting station is predetermined and is loaded into the intelligent system. As soon as the train reaches the next station, the counter detects that the train has crossed x km, that is it has reached the next station and automatically send a text message to the receiver in the station. The text message will contain data such as its arrival time, departure time, train number and next destination. There is WAP to HTTP converter used here that uploads the data into the server. As the distance from one station to the consecutive station where the train stops is known and is loaded into the memory of the system inside the train, it keeps sending text messages to the receivers present in the stations in this manner.

In case there is a delay and train has to halt in an unknown location, there will be a time counter embedded into the receivers fixed in the stations that keeps a track of the trains arriving and leaving. When a train is delayed at a station or outskirt area, the next station where the train has a stopover automatically acknowledges the delay as there is a timer mechanism inside the receiver. Once the receiver does not receive a message from the train at that particular predetermined time it automatically send a message by itself to the server stating the train delay. The information send by the receiver to the server will contain the time by which the train has been delayed, expected arrival time based on its distance and speed, train number.

Chapter 4

WAP TO HTTP CONVERSION

Topic: **Using Nokia Mobile Web Browser and Nokia WAP gateway simulator to specify the location of a particular transmitter.**

Platforms Used: **Nokia Mobile Web Browser 4.0 and Nokia WAP Gateway Simulator 4.0.**

Nokia Mobile Web Browser 4.0:

Theory: In general, Nokia Mobile Web Browser is a development tool intended for mobile Internet content developers who wish to preview how their content will look before it is ultimately deployed on a mobile handset. Using NMB, content developers can display any mobile internet content using Nokia Mobile Internet Toolkit 4.0, as well as local file content and content resident on Internet servers and accessed through WAP connection. WAP connections may be made through either a WAP gateway server, Nokia Mobile browser uses the Nokia Mobile Browser software developed by Nokia. It is used to display an extensive range of current and evolving technologies of internet to mobile internet developers. The transmitter sends a message that is received by a GSM mobile station. The message received by the base station will also have the location of the message transfer. In this way the location of the transmitter can be figured out. In this way the message is loaded onto the internet.

The message on the internet is accessed by the Nokia Mobile Web Browser by loading the specific URL.

The URL loaded can specify any information taken from a particular website or any stored information in hard disk. The address of the path (or website) is actually written in the Load URL menu.

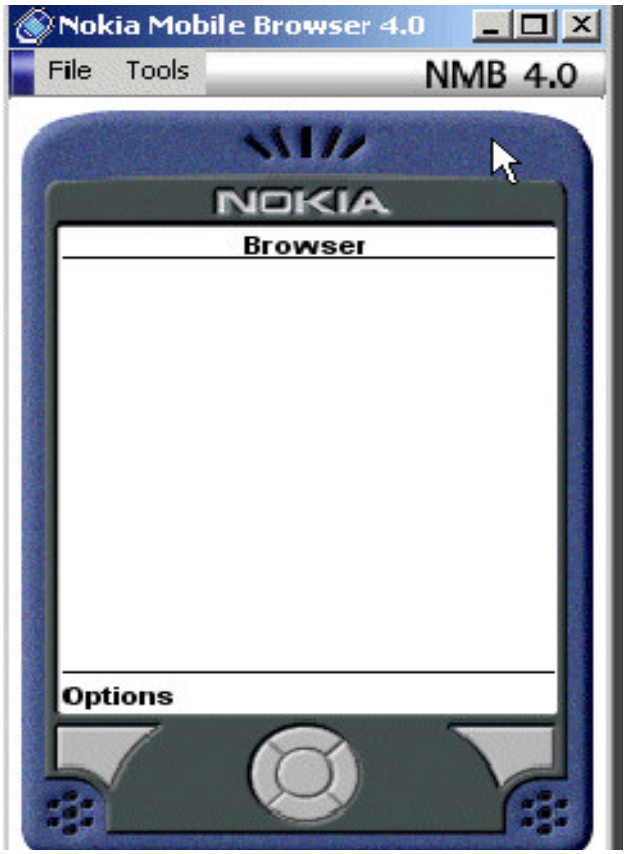

Fig.4.1: Nokia Mobile Web Browser

Steps:

Loading URL: Choose the Load URL menu item to load content from the internet. Nokia WAP Gateway Simulator should be selected before loading the URL.

After installing NMB, the configuration is done using the Settings dialogue through the Tools > Settings menu item.

- Configure NMB to use Nokia WAP Gateway Simulator. By default NMB is configured to use NWGS whose I.P. is 127.0.0.1.
- As we are using NWGS as the WAP gateway, NWGS should be launched in advance.

Now the loaded URL provides the information regarding the location, time of transmission and other details of the transmitter.

Nokia WAP Gateway Simulator 4.0:

Wireless Application Protocol is an open international standard for the applications that uses wireless communication. Its main purpose is to access the internet from a mobile phone or PDA. A WAP browser provides all the basic services of a computer based Web browser but simplified to operate within the restriction of mobile phone.

A WAP gateway sits between mobile devices using the WAP protocol and the World Wide Web. This translates pages into a form suitable for the mobiles. NWGS is a single user WAP gateway simulator based on the multi user Nokia Active server. It comes with a decoder for decoding incoming requests from WAP client user agents so that these can be forwarded over the HTTP protocol over Internet servers. It also includes encoders that do the vice versa.

In the NWGS, the top window is the administration window that is used to configure and manage the server. The bottom window represents the running server display.

NWGS is meant for single user running development programs. So as long as NWGS is running, a phone or other application development program may be used to connect to the internet.

The data received is stored in a Microsoft access database.

We use a Baby Web server, which is an HTTP server provider, so a WAP to HTTP converter NWGS 4.0 is used. This simulator converts WAP to HTTP format. After receiving the data from the transmitter (NWGS), it updates the information on the database.

Java Database Connectivity is used to store the information into the database.

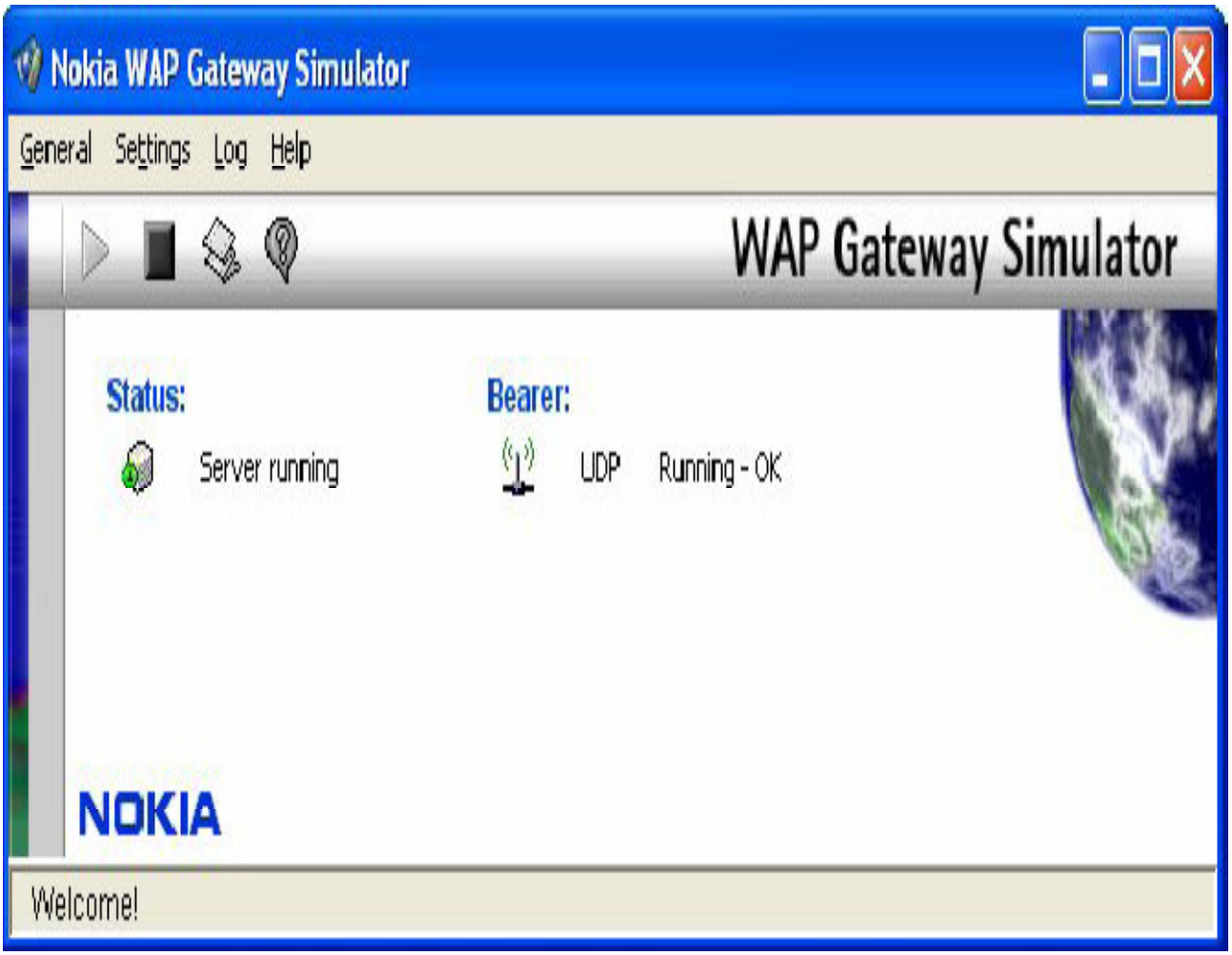

Fig.4.2: Software window of the NWGS.

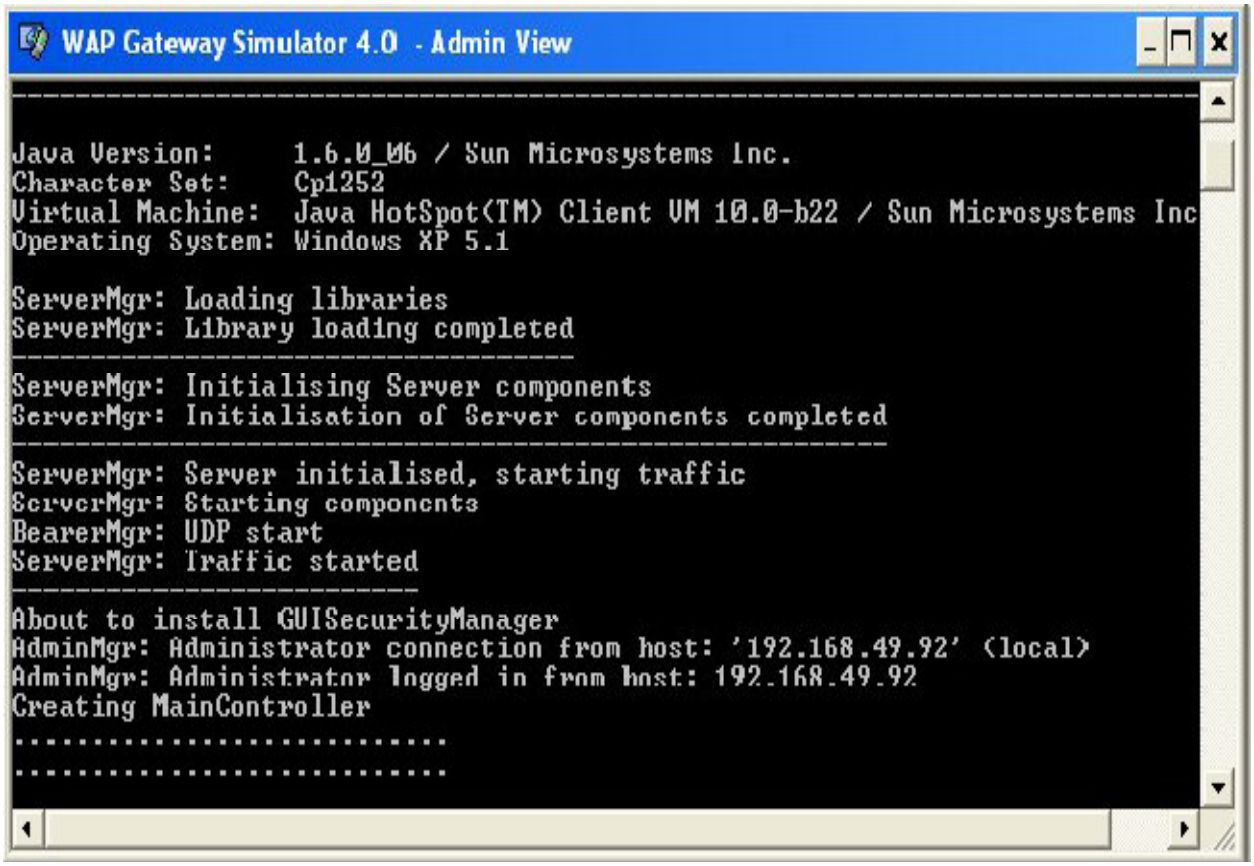

Fig.4.3: Administrator window of NWGS.

Chapter 5

ESTABLISHMENT OF CONNECTION BETWEEN NWGS AND DATABASE

Java code for establishing connection between MS Access Database and WAP Gateway simulator.

Connection code for connecting Access Database to NWGS:

```
class for Name ("Sun.jdbc.odbc.Jdbc Odbc Driver");

connection= Driver Manager.get connection("jdbc.odbc.Miller");

stmt= connection.create statement(Result Set.TYPE_SCROLL_SENSITIVE, Result Set.CONCUR_UPDATABLE);

sql Query= "Select Train_no, Train_name, Train_codeno, Arrival_time,Dep_time";

rs= stmt.execute query;
```

Code for inserting the data:

```
class.insert button handler implements Action Listener

{

public void action performed(Action Event e)

{
```

```
try
{
if (inserting == false)
{ // clear text boxes
numF.Set Text(“ “);
trainF.Set Text(“ “);
codeno.Set Text(“ “);
arrival.Set Text(“ “);
dep.Set Text(“ “);
insert Button.Set Text(“Save“);
inserting= True;
}
else
{ // finalise the insert
rs.Move to Insert row();
int  Train_no= Integer.passe Int(numF.get text());
rs.update Int(1, Train_no);
rs.update string(2, trainF.get text());
rs.update string(3, codeno.get text());
rs.update string(4, arrival.get text());
rs.update string(5, dep.get text());
rs.insert row();
```

```
insert Button.set text(“Insert”);

inserting= False;

}

}

catch (Exception ex)

{

Joption Parse.show message Dialog (null, “Nothing Inserted”, “Event Handler Message”);

Joption Parse.Information_Message;

}

}

}
```

Chapter 6

CREATION OF LIVE RAILWAY TIME-TABLE

Interfacing Ms-Access database with Netbeans in order to generate Live Railway Timetable

Aim: A railway time-table/database is created using Netbeans, Ms-Access. The time-table displays train arrival time, departure time, train number, train name and the time table can be updated using JDBC programming. Information can be retrieved and loaded into it.

Theory: Ms-Access is used to create a database. The database will contain information in the form of rows and columns. Say it will have six rows and four columns. Each column will represent the respective train number, train time and the new updated train arrival or departure time.

In order to generate the client, server and the train information for display Netbeans is used. Each one of them is programmed with a separate JAVA program.

Steps: The database is generated. We have used four columns depicting train number, train name, train time, update train time. Six trains have been taken for usage in the database with their corresponding train numbers, arrival time etc. the number of trains can be extended to as many as trains required. For simulation purpose we have used six trains

.

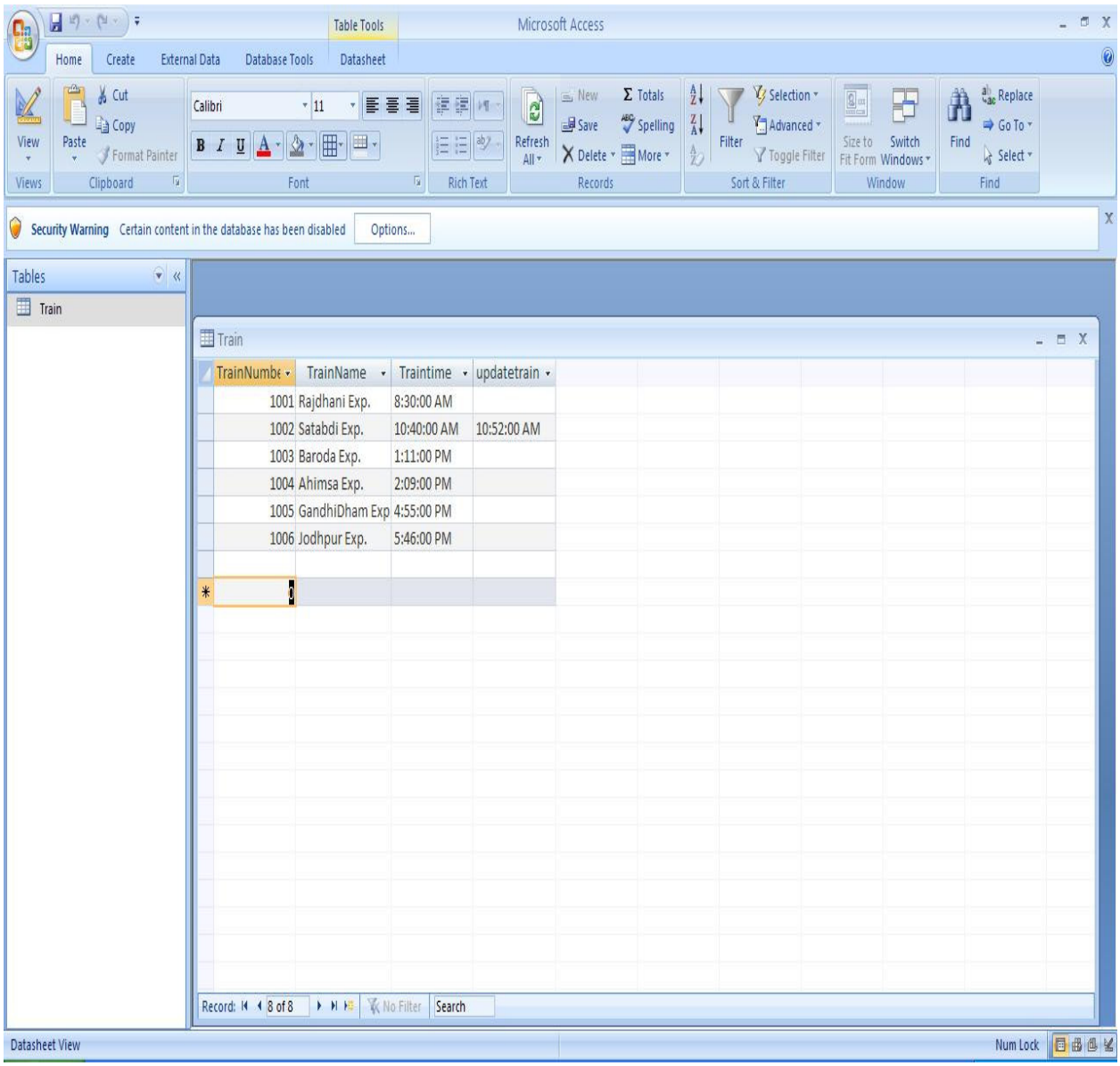

Fig.6.1: Microsoft Access Database.

NETBEANS IDE 6.7 M2:

Netbeans is used in order to generate the client, server and the train information window using JAVA programming. These are the main programming components of a Train time information system.

Three different programs are used to generate the three different windows. These JAVA programs were made in such a manner that they were interfaced with Ms-Access database.

JAVA programs for SERVER, CLIENT, TRAIN INFORMATION:

Program 1: SERVER

```
package ATO;

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

import javax.swing.event.*;

import java.io.*;

import java.util.*;

import java.net.*;

import java.sql.*;
```

```
public class Server extends JFrame implements ActionListener{

        private ObjectInputStream ois = null;
        private ObjectOutputStream oos = null;
        private ServerSocket ss = null;

        private String msg="";

        private Statement stmt = null;
        private ResultSet rs = null;
        private Connection con = null;

        private JLabel trainnolbl = new JLabel("Train No.");
        private JTextField trainnotxt = new JTextField(4);
        private JPanel trainnopnl = new JPanel();
        private JLabel trainnamelbl = new JLabel("Train Name:");
        private JTextField trainnametxt = new JTextField(20);
        private JPanel trainnamepnl = new JPanel();
        private JLabel traintimelbl = new JLabel("Train Time:");
        private JTextField traintimetxt = new JTextField(10);
        private JPanel traintimepnl = new JPanel();
```

```
private JButton defaultbtn = new JButton("Default Time");
private JButton searchbtn = new JButton("Search Train");
private JButton updatebtn = new JButton("Update");
private JPanel btnpnl = new JPanel();

private Vector allports = new Vector();
private Vector allips = new Vector();


private JPanel allpnl = new JPanel();
public Server()
{
        super("ATO WiMAX Railway Server");
        trainnopnl.add(trainnolbl);
        trainnopnl.add(trainnotxt);
        trainnamepnl.add(trainnamelbl);
        trainnamepnl.add(trainnametxt);
        traintimepnl.add(traintimelbl);
        traintimepnl.add(traintimetxt);
        btnpnl.add(searchbtn);
        btnpnl.add(updatebtn);
        btnpnl.add(defaultbtn);
```

```
allpnl.setLayout(new GridLayout(4,2));
allpnl.add(trainnopnl);
allpnl.add(trainnamepnl);
allpnl.add(traintimepnl);
allpnl.add(btnpnl);
this.getContentPane().add(allpnl);
this.addWindowListener(new WindowAdapter()
                {
                        public void windowClosing()
                        {
                                System.exit(0);
                        }
        });

this.setSize(400,400);
this.setVisible(true);
updatebtn.addActionListener(this);
searchbtn.addActionListener(this);
defaultbtn.addActionListener(this);
try
{
```

```
        ss = new ServerSocket(20000);
        }
        catch(IOException ie)
        {}
        serveraccept sra = new serveraccept();
        sra.start();
}

public class serveraccept extends Thread
{
        private Socket s=null;
        public void run()
        {
                while(true)
                {
                        try
                        {
                                s = ss.accept();
                                ois = new ObjectInputStream(s.getInputStream());
                                TrainInfo tif = (TrainInfo)ois.readObject();
                                allports.addElement(new Integer(tif.getPortno()));
                                allips.addElement(s.getInetAddress());
```

```
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    con =
DriverManager.getConnection("jdbc:odbc:Train","","");
                    stmt = con.createStatement();

                    String qry = "select * from Train";

                    rs = stmt.executeQuery(qry);

                    while(rs.next())
                    {
                        msg += "***Train No.->" + rs.getString(1) + "  Train
Name:->" + rs.getString(2) + "  Train Time:->" + rs.getString(3) + "***" ;
                    }

                        //InetAddress conip =
(InetAddress)allips.elementAt(i);
                        //int pt = (Integer)allports.elementAt(i);
                        //Socket serve = new Socket(conip,pt);
                        oos = new
ObjectOutputStream(s.getOutputStream());
```

```
                    oos.writeObject(new String(msg));



                } // end try
                catch(IOException ie)
                {
                    System.out.println("IO Exception in ServerAccept:
" + ie);
                }
                catch(ClassNotFoundException cnfe)
                {
                    System.out.println("ClassNotFound Exception in
ServerAccept: " + cnfe);
                }
                catch(SQLException sqle)
                {
                    System.out.println("IO Exception in ServerAccept:
" + sqle);
                }
            }// end while
        }// end run
    } // end thread
```

```
        public void actionPerformed(ActionEvent ae)
        {
                if(ae.getSource() == searchbtn)
                {
String qry = "select * from Train where TrainNumber=" + trainnotxt.getText();
                        try
                        {
                                rs = stmt.executeQuery(qry);
                                if (! rs.next())
                                        JOptionPane.showMessageDialog(null,"No Such
Train Exists","Failed",JOptionPane.ERROR_MESSAGE);
                                else
                                {

                                        trainnotxt.setText(rs.getString(1));
                                        trainnametxt.setText(rs.getString(2));
                                        traintimetxt.setText(rs.getString(3));
                                }
                        }
                        catch(SQLException sqle)
                        {
```

```
                    System.out.println("SQL Exception in ActionEvent:" +
sqle);
                }
                }
            if(ae.getSource() == updatebtn)
            {
String qry = "update Train set updatetrain='" + traintimetxt.getText() + "' where
TrainNumber=" + Integer.parseInt(trainnotxt.getText());
                try
                {
                     stmt.executeUpdate(qry);
                }
                catch(SQLException sqle)
                {
                    System.out.println("SQL exception in update:" + sqle);
                }
                String qrey = "select * from Train";
try
                {
              rs = stmt.executeQuery(qrey);
               msg ="";
              while(rs.next())
```

```
                {
msg += "***Train No.->" + rs.getString(1) + "  Train Name:->" + rs.getString(2) + "
Train Time:->" + rs.getString(4) + "***" ;
                }
                    }
                    catch(SQLException sqle)
                    {
                System.out.println("sql exception in forming msg:" + sqle);
                    }

                    for(int i=0;i<allips.size();i++)
                    {
                        try
                        {
                        InetAddress conip = (InetAddress)allips.elementAt(i);
                        int pt = (Integer)allports.elementAt(i);
                        Socket serve = new Socket(conip,pt);
                        oos = new
ObjectOutputStream(serve.getOutputStream());
                        oos.writeObject(new String(msg));
                        }
                        catch(IOException ie)
```

```
                    {
System.out.println("IO exception in sending msg to all clients: " + ie);
                    }
                }
            }// end if

            if(ae.getSource() == defaultbtn)
            {
                String qry = "select * from Train";
                  try
                  {
                      rs = stmt.executeQuery(qry);
                      msg="";
                      while(rs.next())
                      {
msg += "***Train No.->" + rs.getString(1) + "  Train Name:->" + rs.getString(2) + "
Train Time:->" + rs.getString(3) + "***" ;
                      }
                  }
                  catch(SQLException sqle)
                      {
            System.out.println("sql exception in forming msg:" + sqle);
```

```
                    }
                for(int i=0;i<allips.size();i++)
                    {
                        try
                        {
                       InetAddress conip = (InetAddress)allips.elementAt(i);
                        int pt = (Integer)allports.elementAt(i);
                        Socket serve = new Socket(conip,pt);
                        oos = new
ObjectOutputStream(serve.getOutputStream());
                        oos.writeObject(new String(msg));
                        }
                        catch(IOException ie)
                        {
        System.out.println("IO exception in sending msg to all clients: " + ie);
                        }
                    }
}

        }// end actionperformed
        public static void main(String []args)
        {
```

```
            new Server();
        }
}
```

Program 2: CLIENT

```
package ATO;


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.io.*;
import java.util.*;
import java.net.*;
public class Client extends JFrame {


        private JLabel trainmsg = new JLabel("");
        private JPanel trainmsgpnl = new JPanel();
        private Socket s = null;
        private int portno = 20000;
        private String msg = "";
        private ObjectInputStream ois = null;
```

```
private ObjectOutputStream oos = null;
private ServerSocket ss = null;
private int p=20001;
public Client()
{
	super("ATO WiMAX Railway Client");
	//trainmsgpnl.setLayout(new GridLayout(1,1));
	//trainmsgpnl.add(trainmsg);
	this.getContentPane().add(trainmsg);
	this.addWindowListener(new WindowAdapter()
	{
		public void windowClosing()
		{
			System.exit(0);
		}
	});
	this.setSize(400,200);
	this.setVisible(true);
	try
	{
	ss = new ServerSocket(++p);
	}
```

```
catch(IOException ie)
{
        for(;;)
        {
        p++;
                try
                {
                ss = new ServerSocket(p);
                break;
                }
                catch(IOException ioe)
                {}
        }
}
TrainInfo tif = new TrainInfo();
tif.SetPortno(p);
        try
        {
                s = new Socket("127.0.0.1",portno);
                oos = new ObjectOutputStream(s.getOutputStream());
                oos.writeObject(tif);
                ois = new ObjectInputStream(s.getInputStream());
```

```
                    try{
                    msg = (String)ois.readObject();
                    }
                    catch(ClassNotFoundException cnfe)
                    {
                    System.out.println("class not found Wrapper:" + cnfe);
                    }
                    ReadMsg rmg = new ReadMsg();
                    rmg.start();
                    Roll rl = new Roll();
                    rl.start();
}
                catch(IOException ie)
                {}
}

public class ReadMsg extends Thread
{
        private Socket s=null;


        public void run()
```

```
        {
                while(true)
                {
                        try
                        {
                                s = ss.accept();
                                ois = new ObjectInputStream(s.getInputStream());
                                msg = (String)ois.readObject().toString();
                        }
                        catch(IOException ie)
                        {}
                        catch(ClassNotFoundException cnfe)
                        {}
                }
        }
}

public class Roll extends Thread
{
        public void run()
        {
                while(true)
```

```
                {
                        try{
                        Thread.sleep(250);
                        }
                        catch(InterruptedException iet)
                        {}
                        msg = msg.substring(1) + msg.charAt(0);
                        trainmsg.setText(msg);
                }

        }
}

public static void main(String []args)
{
        new Client();
}}
```

Program 3: TRAIN INFORMATION DISPLAY

```
package ATO;


import java.io.*;
import java.util.*;
public class TrainInfo implements Serializable{
        private int trainno = 0;
        private String trainname="";
        private String time = "";
        private int portno=0;
        public TrainInfo()
        {
        }
        public void SetPortno(int portno)
        {
                this.portno = portno;
        }
        public int getPortno()
        {
return portno; }}
```

After entering the respective programs in Netbeans, on the left end of Netbeans interface we will have the Project toolbar. Clicking there will give us the ability to run all three programs. After running them we will get a window open that will ask us to enter the train name, train number, and update time. Upon entering the details and pressing OK, a new window will open up showing the corresponding train details.

In case if the train is arriving late, it's new time can be loaded in the update time slot and then automatically the new time will be uploaded into the database.

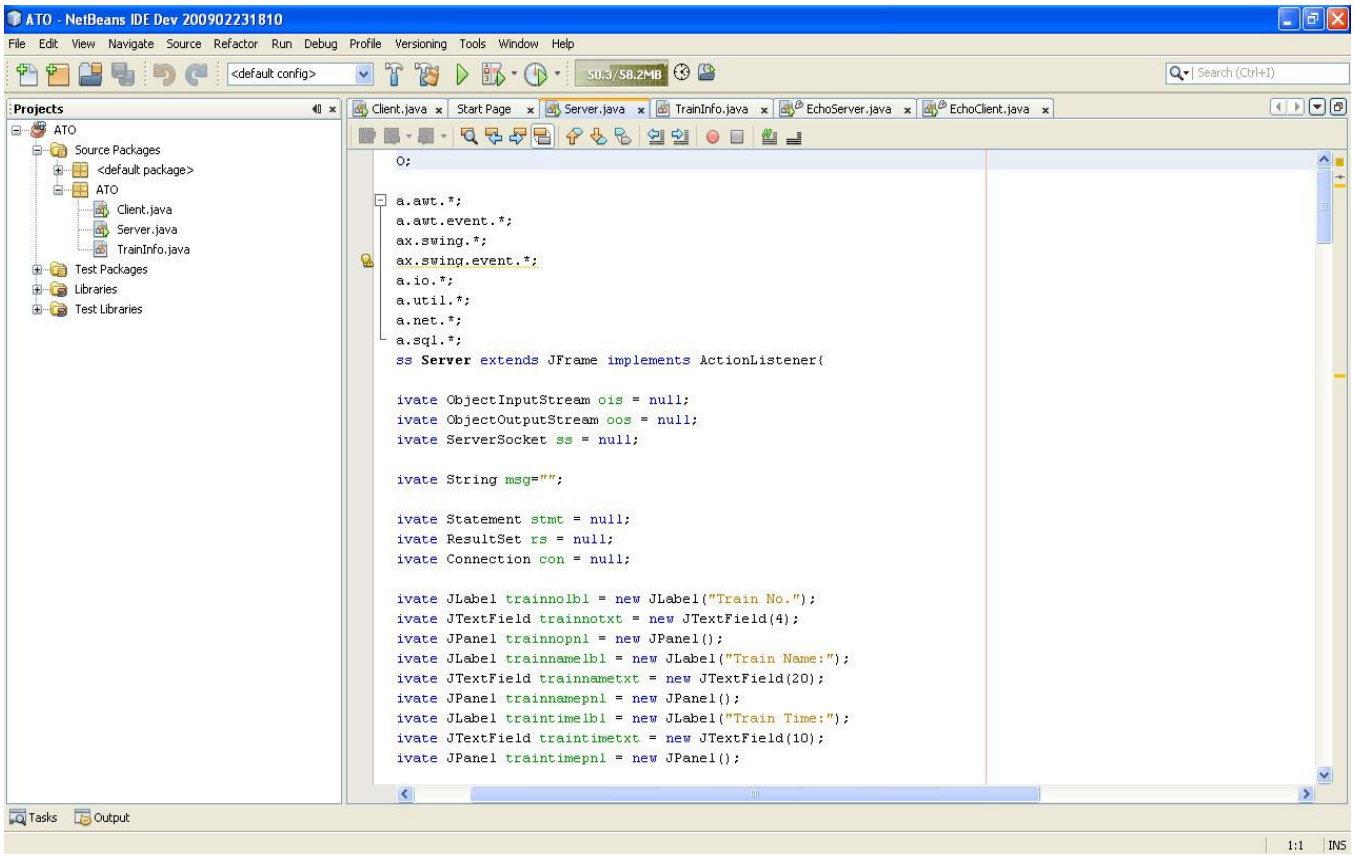

Fig.6.2: Server program in Netbeans IDE.

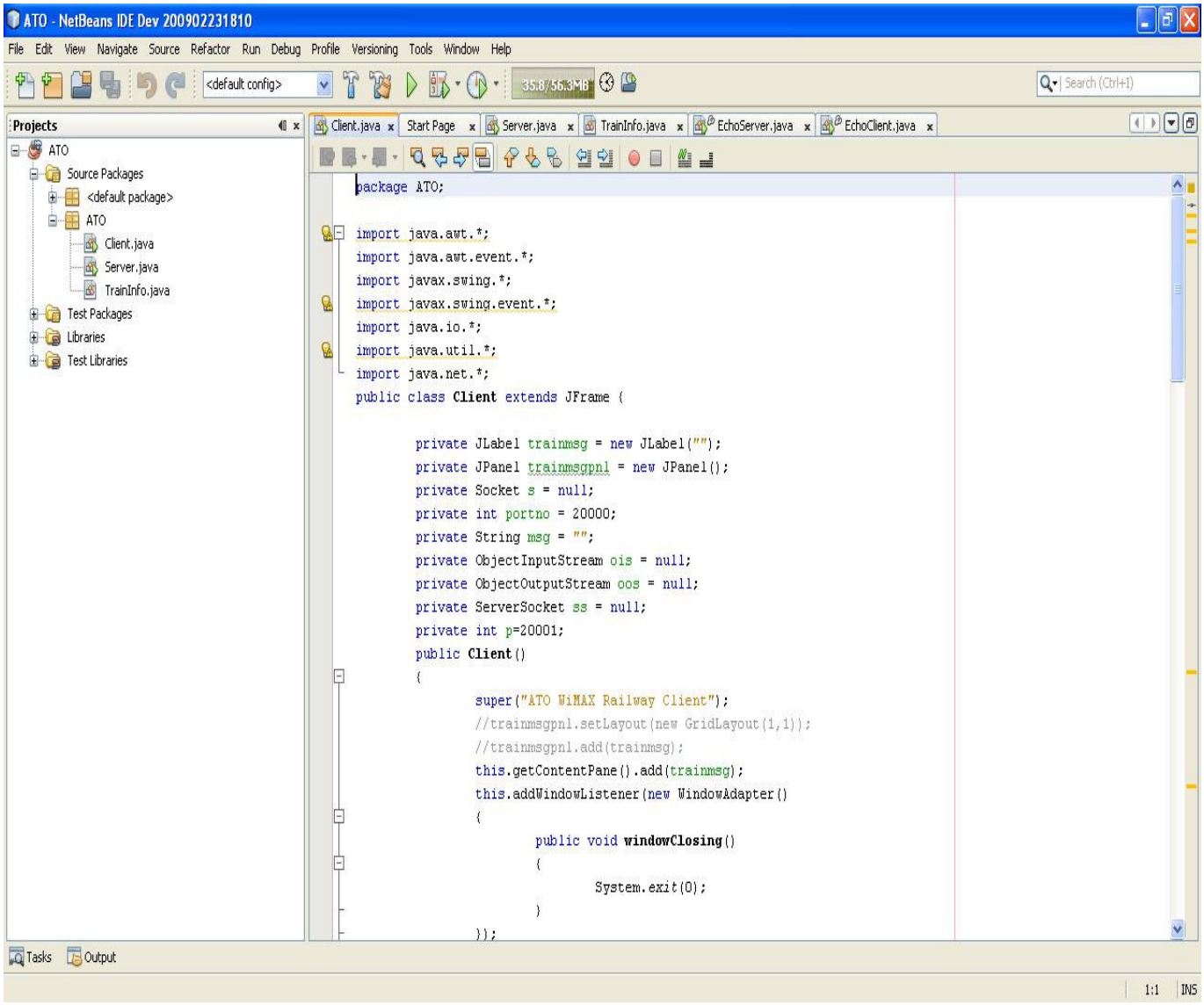

Fig.6.3: Client program in Netbeans IDE.

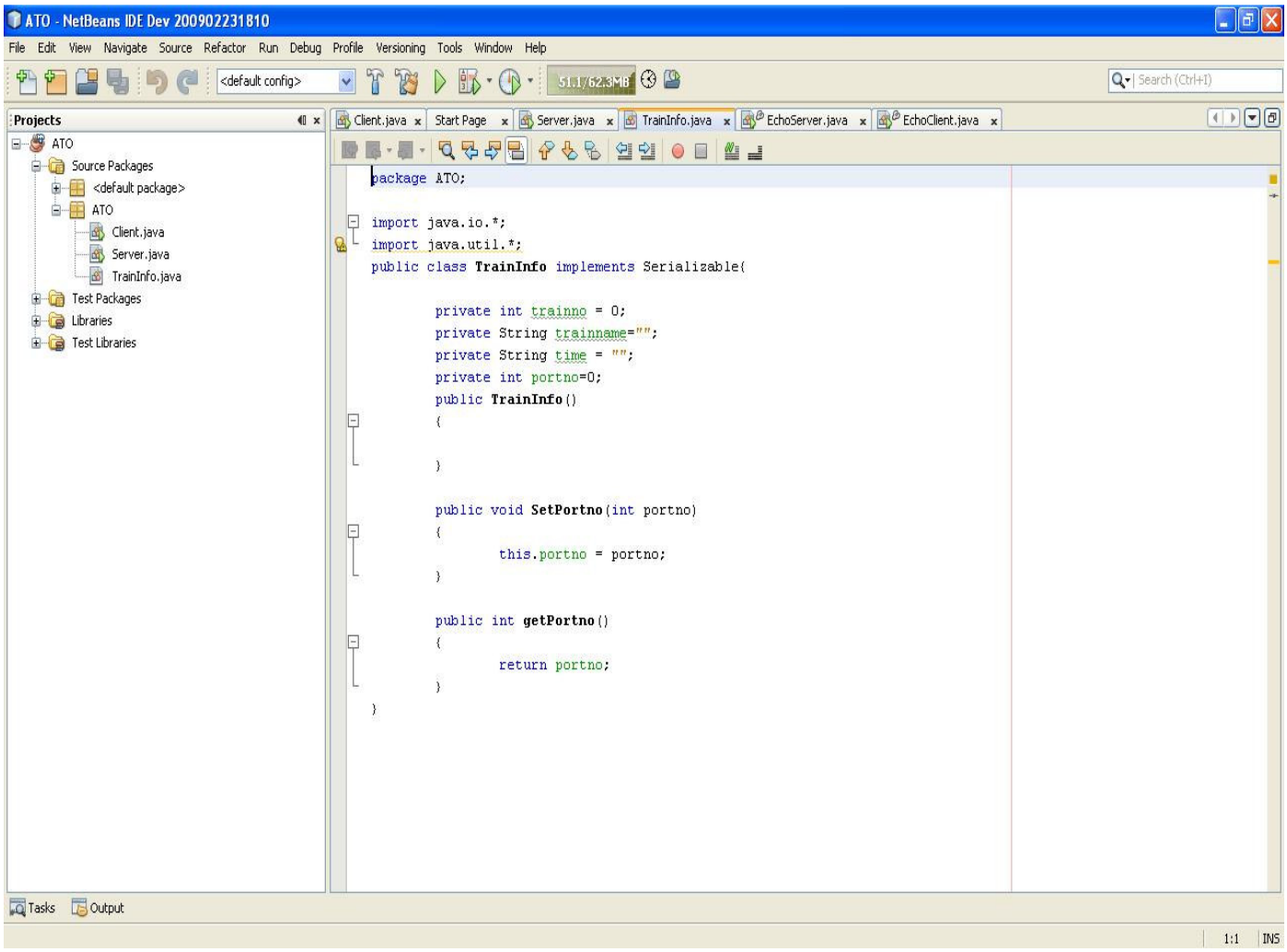

Fig.6.4: Program for Train Information in Netbeans IDE.

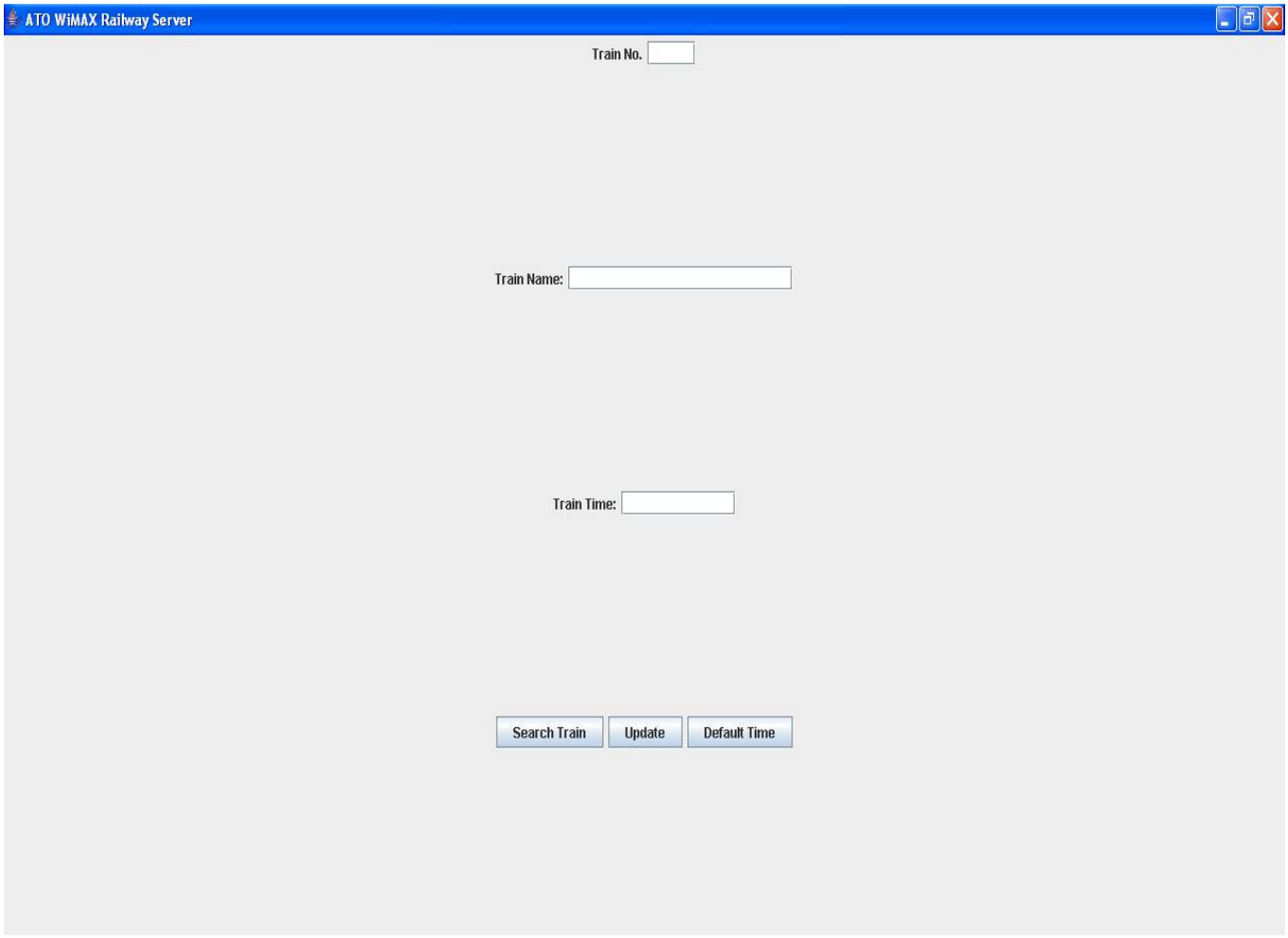

Fig.6.5: Generated Railway Server.

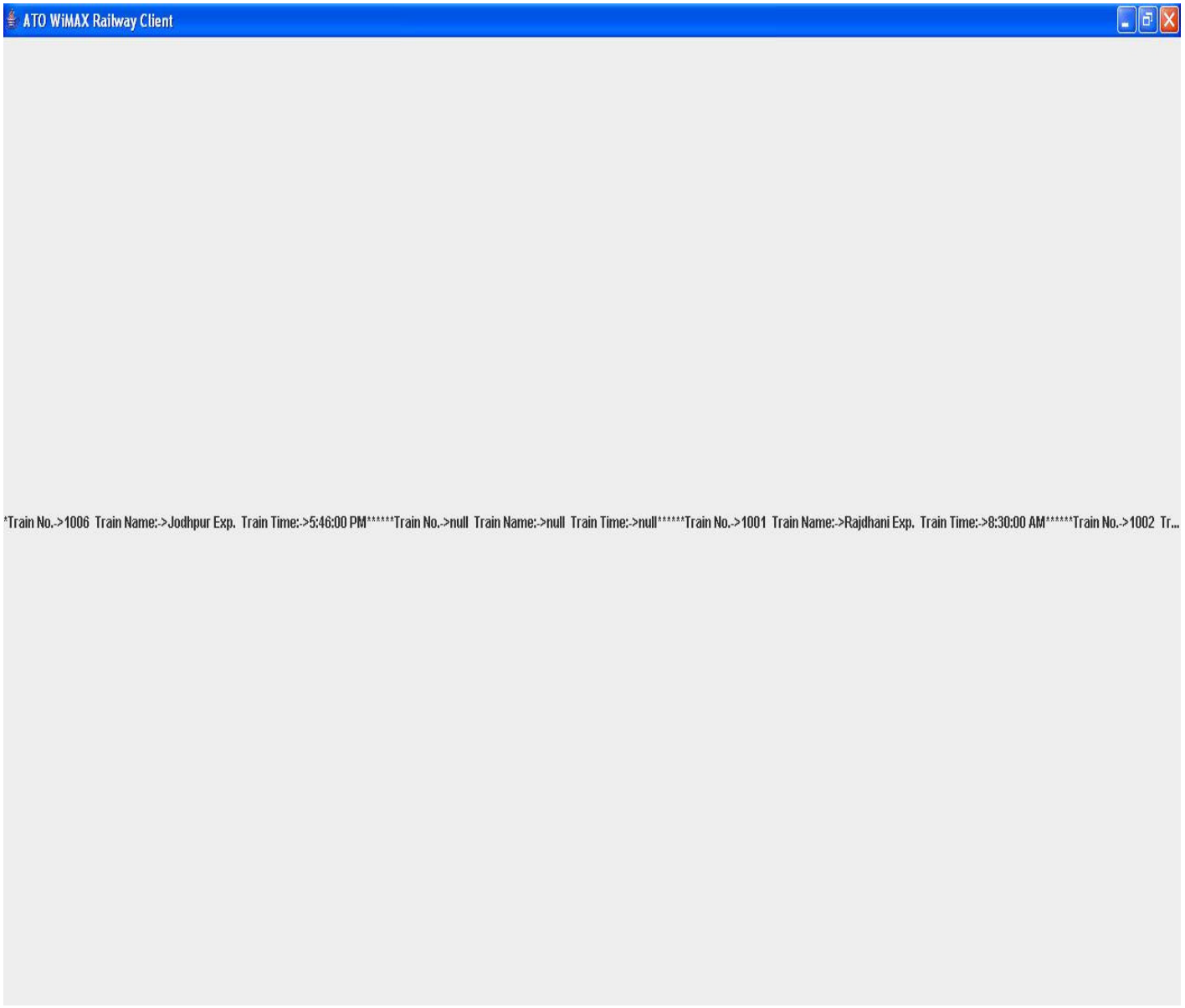

Fig.6.6: Generated Railway Client Service.

Chapter 7

INTERFACE OF DATABASE WITH MULTIPLE TRAINS

INTERFACING THE DATABASE WITH SMS'S FROM MULTIPLE MOBILES, IN TURN USED TO RETRIEVE INFORMATION OR UPDATE INFORMATION ONTO THE DATABASE

Platform: **SMSLibX , Ms-Access**

Theory: An existing Access database is used here. Create a new VBA form, then switch to the VBA source code (view>structure, then view> code menu).

Setup reference to SMSLibX library (Tools > Reference menu)

Step: The VBA source code can be used into an Access form in order to send and receive SMS messages.

```
' Declare SMSModem object
Public WithEvents Modem As SMSModem

' Send SMS
Private Sub cmdSendMessage_Click()

  ' Open modem communication
  Set Modem = New SMSModem
  Modem.LogTrace = True
  Modem.OpenComm ModemType, ModemPort, , smsNotifyAll

  ' Send message
  Call Modem.SendTextMessage(PhoneNumber, MessageText)
End Sub

' Receive SMS by event
Private Sub Modem_MessageReceived(Message As SMSLibX.SMSDeliver)
  MsgBox "New message received from " & Message.Originator & ":" _
```

```
    & vbCrLf & vbCrLf & Message.Body, _
    vbInformation, "New message received"
End Sub
```

The GSM modem-SIMADO is used here. The messages are received by it from multiple mobile phones and it is converted from WAP TO HTTP format, from where it gets loaded onto the database with the help of SMSLibX.

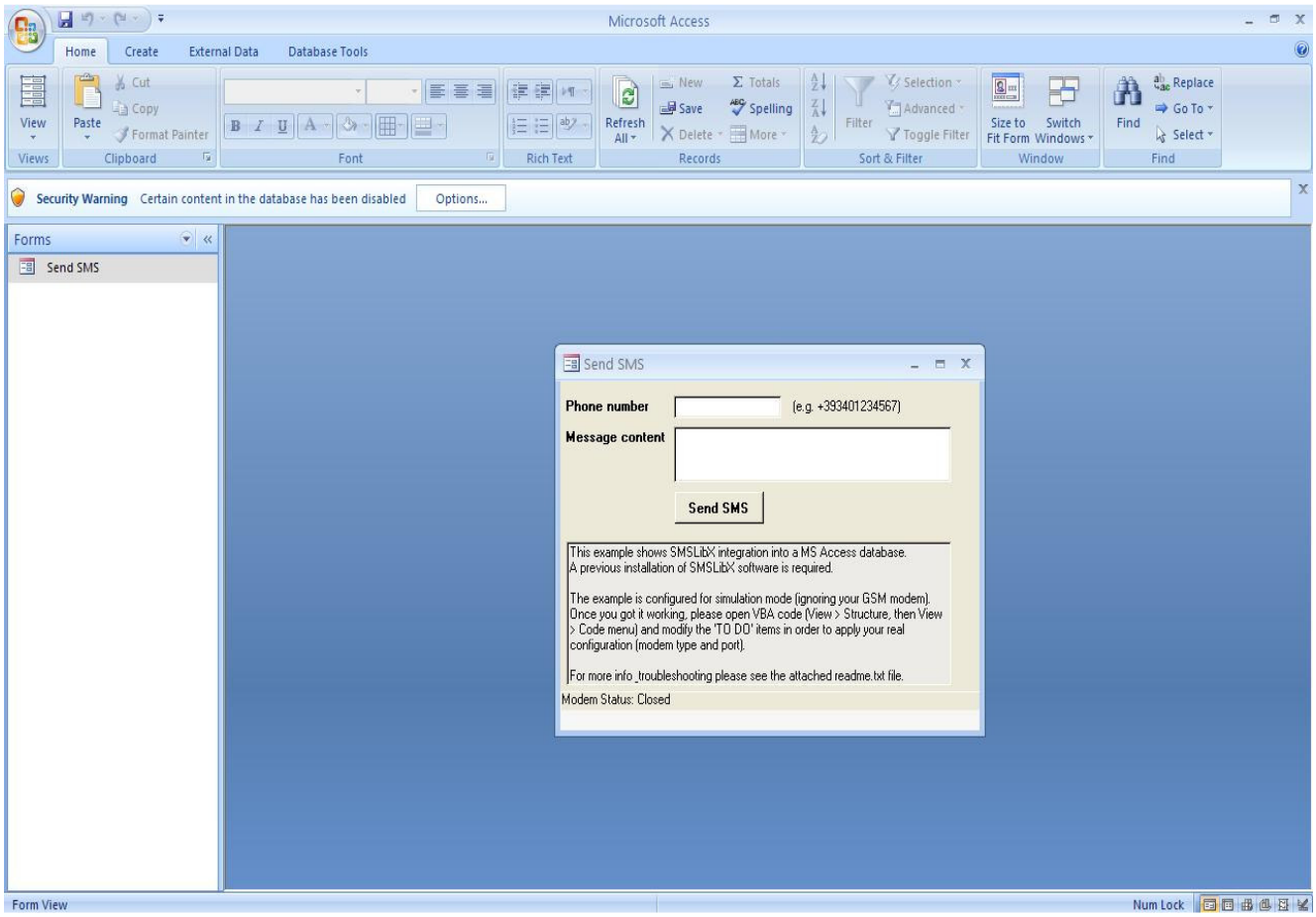

Fig.7.1: MS Access Database interfaced with SMSLibX.

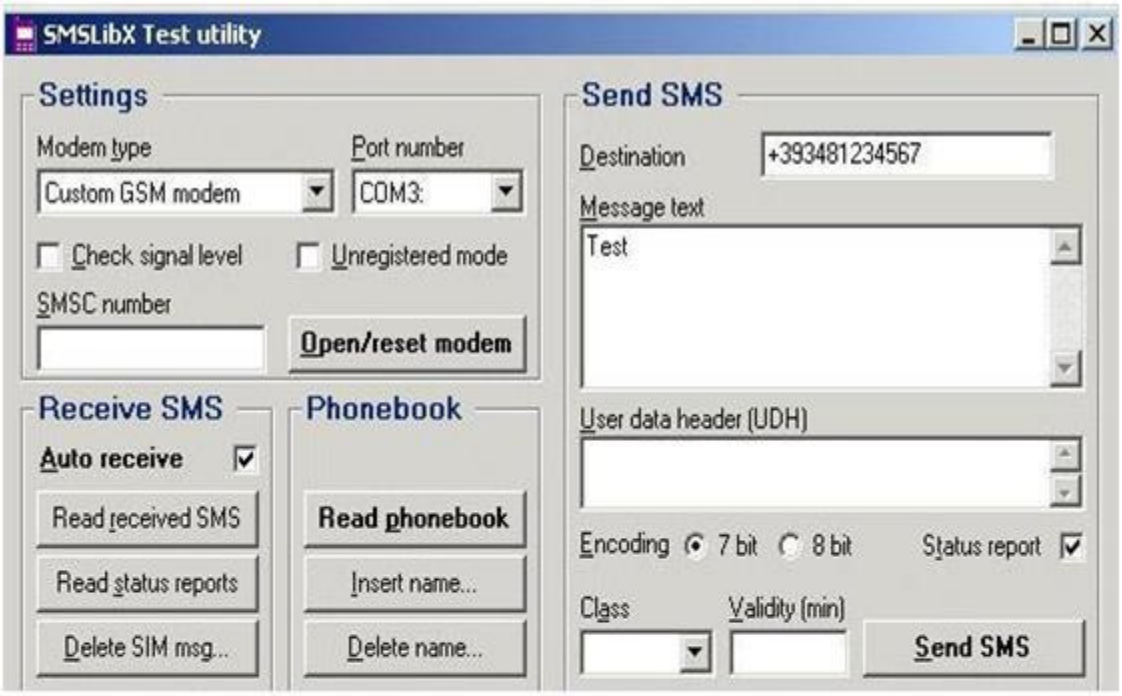

Fig 7.2: SMSLibX Window

CONCLUSION AND FUTURE SCOPE:

The automatic Train running information system has been implemented using Qualnet, Netbeans, Nokia Mobile Browser, Nokia WAP Gateway Simulator and SMSLibX softwares. The results for each simulation were satisfactory. Many developments could be done in future to implement the system on commercial scale. The system could be interfaced with a master clock which governs many other slave clocks attached in each platform. This provision could make automatic platform allotment possible. The system then could provide proper signaling facilities in the station.

REFERENCES:

1. **Qualnet Simulator manual**
2. **Netbeans IDE manual**
3. **SIMADO Modem manual**
4. **Nokia Mobile Browser manual**
5. **Nokia WAP Gate Simulator manual**
6. **SMSLibX manual**
7. **Online References:**
    - **http://www.smsco.it/tomcat/en/sms_tutorials/sms_from_access.jsp**
    - **http://www.functionx.com/vbaccess/Lesson01.htm**
    - **http://www.forum.nokia.com/**